Europäisches Patentamt
European Patent Office
Office européen des brevets

Bescheinigung | Certificate | Attestation

Die angehefteten Unterlagen stimmen mit der ursprünglich eingereichten Fassung der auf dem nächsten Blatt bezeichneten europäischen Patentanmeldung überein.

The attached documents are exact copies of the European patent application described on the following page, as originally filed.

Les documents fixés à cette attestation sont conformes à la version initialement déposée de la demande de brevet européen spécifiée à la page suivante.

Patentanmeldung Nr. Patent application No. Demande de brevet n°

03019428.6

Der Präsident des Europäischen Patentamts;
Im Auftrag

For the President of the European Patent Office

Le Président de l'Office européen des brevets
p.o.

R C van Dijk

Europäisches Patentamt | European Patent Office | Office européen des brevets

Anmeldung Nr:
Application no.: 03019428.6
Demande no:

Anmeldetag:
Date of filing: 28.08.03
Date de dépôt:

Anmelder/Applicant(s)/Demandeur(s):

PACT XPP Technologies AG
Muthmannstrasse 1
80939 München
ALLEMAGNE

Bezeichnung der Erfindung/Title of the invention/Titre de l'invention:
(Falls die Bezeichnung der Erfindung nicht angegeben ist, siehe Beschreibung.
If no title is shown please refer to the description.
Si aucun titre n'est indiqué se referer à la description.)

Device and method for data processing

In Anspruch genommene Priorität(en) / Priority(ies) claimed /Priorité(s) revendiquée(s)
Staat/Tag/Aktenzeichen/State/Date/File no./Pays/Date/Numéro de dépôt:

Internationale Patentklassifikation/International Patent Classification/
Classification internationale des brevets:

G06F9/00

Am Anmeldetag benannte Vertragstaaten/Contracting states designated at date of filing/Etats contractants désignées lors du dépôt:

AT BE BG CH CY CZ DE DK EE ES FI FR GB GR HU IE IT LU MC NL PT RO SE SI SK TR LI

PACT

1 Overview of changes vs. XPP V2.0

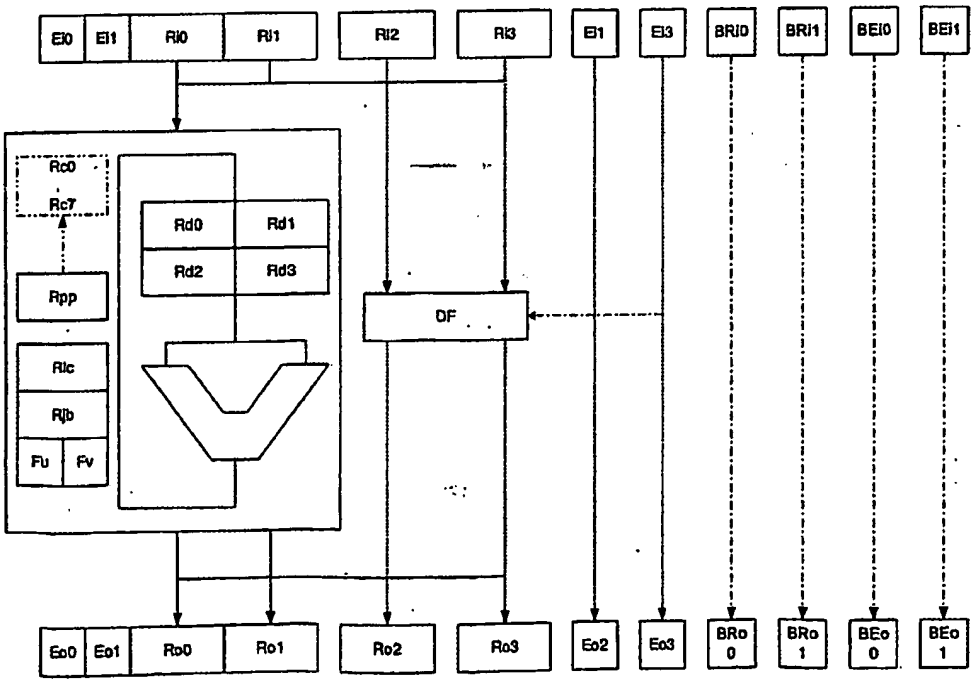
1.1 ALU-PAE Architecture

A PAE comprises 4 input ports and 4 output ports. Embedded with each PAE is the FREG path newly named DF with its dataflow capabilities, like *MERGE*, *SWAP*, *DEMUX* as well as *ELUT*.

2 input ports Ri0 and Ri1 are directly connected to the ALU. Two output ports receive the ALU results.
Ri2 and Ri3 are typically fed to the DF path which output is Ro2 and Ro3.
Alternatively Ri2 and Ri3 can serve as inputs for the ALU as well. This extension is needed to provide a suitable amount of ALU inputs if *Function Folding* (as described later) is used. In this mode Ro2 and Ro3 serve as additional outputs.

Associated to each data register (Ri or Ro) is an event port (Ei or Eo).

It is to decide whether an additional data and event bypass BRi0-1, BEi0-1 is implemented. The decision depends on how often Function Folding will be used and how many inputs and outputs are required in average.



1.1.1 Other extensions

SIMD operation is implemented in the ALUs to support 8 and 16 bit wide data words for i.e. graphics and imaging.

Saturation is supported for ADD/SUB/MUL instructions for i.e. voice, video and imaging algorithms.

1.2 Function Folding

1.2.1 Basics and input/output paradigms

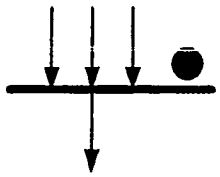
Within this chapter the basic operation paradigms of the XPP architecture are repeated for a better understanding based on Petri-Nets. In addition the Petri-Nets will be enhanced for a better understanding of the subsequently described changes of the current XPP architecture.

Each PAEs operates as a data flow node as defined by Perti-Nets. A Petri-Net supports a calculation of multiple inputs and produces one single output. Special for a Perti-Net is, that the operation is delayed until all inputs are available.

For the XPP technology this means:

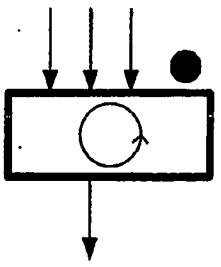
1. all necessary data is available
2. all necessary events are available

The quantity of data and events is defined by the data and control flow, the availability is displayed at runtime by the handshake protocol RDY/ACK.



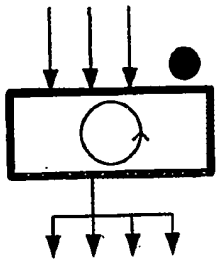
The thick arbor indicates the operation, the dot on the right side indicates that the operation is delayed until all inputs are available.

Enhancing the basic methodology function folding supports multiple operations – maybe even sequential – instead of one, defined as a *Cycle*. Important is, that the basics of Petri-Nets keep unchanged.



Typical PAE-like Petri-Nets consume one input packet per one operation. For sequential operation multiple reads of the same input packet are supported. However, the interface model again keeps unchanged.

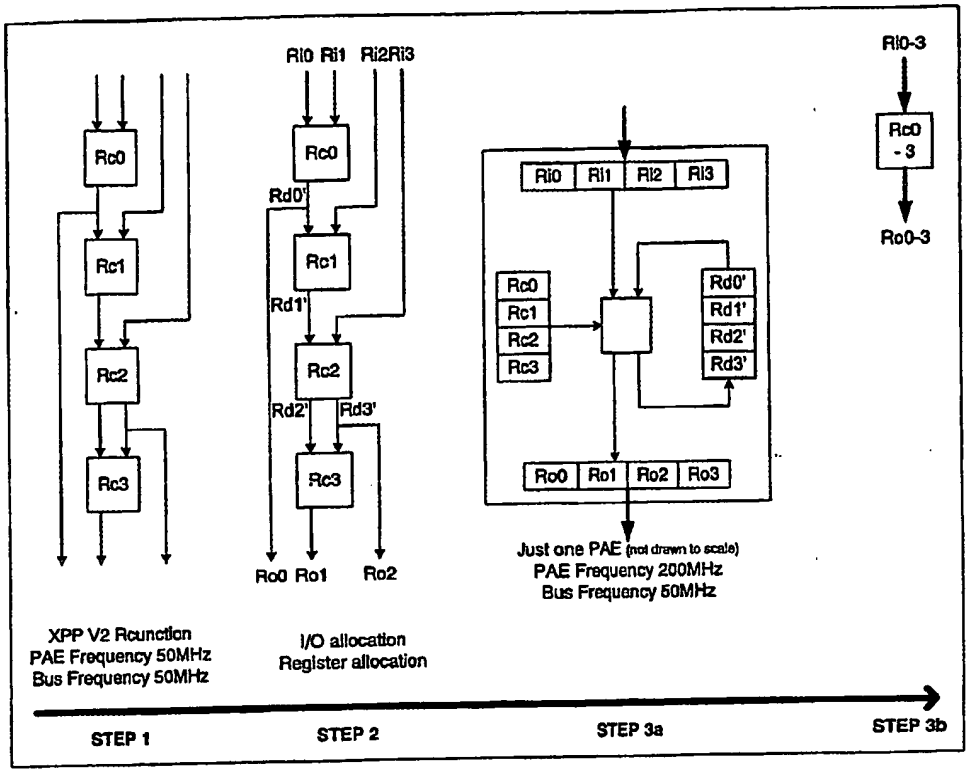
Data duplication occurs in the output path of the Petri-Net, which does not influence the operation basics again.



1.2.2 Method of Function Folding

One of the most important extensions is the capability to fold multiple PAE functions onto on PAE and execute them in a sequential manner. It is important to understand that the intention is not to support sequential processing or even microcontroller capabilities at all. The intention of Function Folding is just to take multiple dataflow operations and map them on a single PAE, using a register structure instead of a network between each function.

The goal is to save silicon area by rising to clock frequency locally in the PAEs. An additional expectation is to save power since the busses operate at a fraction of the clock frequencies of the PAEs. Data transfers over the busses, which consume much power, are reduced.

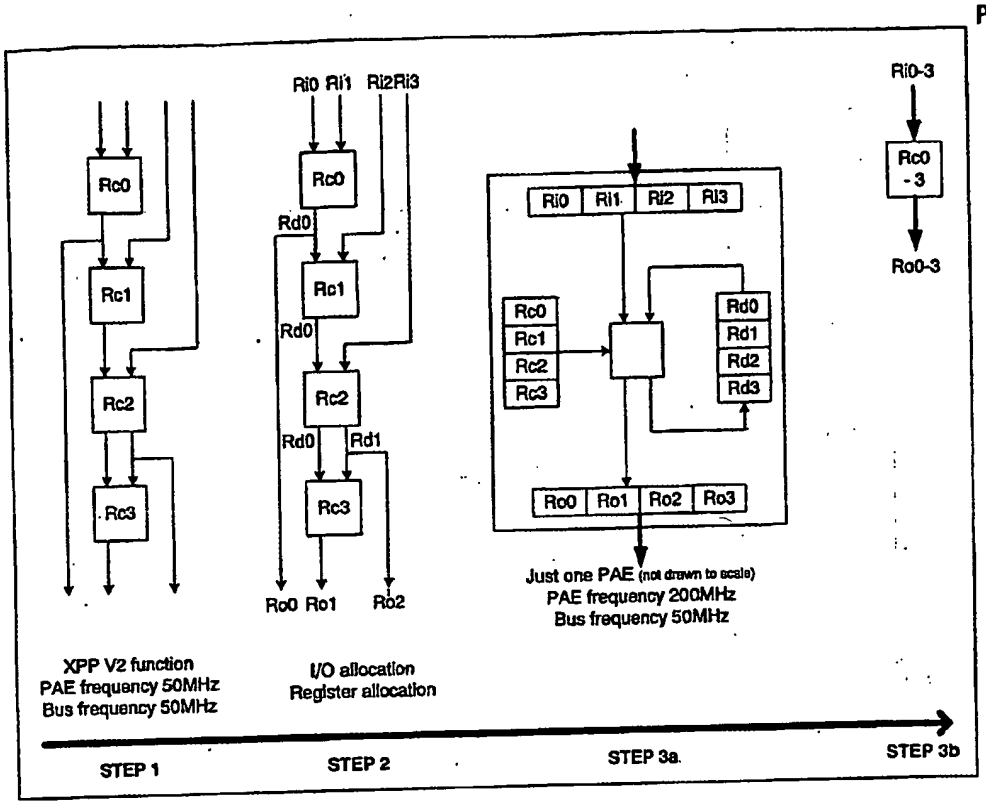


The internal registers can be implemented in two different ways:

1. dataflow model
Each register (r') has a valid bit which is set as soon as data has been written into the register and reset after the data has been read. Data cannot be written if valid is set, data can not be read if valid is not set. This approach implements a 100% compatible dataflow behaviour.
2. sequencer model
The registers have no associated valid bits. The PAE operates as a sequencer, whereas at the edges of the PAE (the bus connects) the paradigm is changed to the XPP-like dataflow behaviour.

Even if at first the dataflow model seems preferable, it has major down sides. One is that a high amount of register is needed to implement each data path and data duplication is quite complicated and not efficient. Another is that sometimes a limited sequential operation simplifies programming and hardware effort.
Therefore it is assumed consecutively that sequencer model is implemented. Since pure dataflow can be folded using automatic tools the programmer should stay within the dataflow paradigm and not be confused with the additional capabilities. Automatic tools must take care i.e. while register allocation that the paradigm is not violated.

The following figure shows that using sequencer model only 2 registers (instead of 4) are required:



For allowing complex function like i.e. address generation as well as algorithms like "IMEC"-like data stream operations the PAE has not only 4 instruction registers implemented but 8, whereas the maximum bus-clock vs. PAE-clock ration is limited to a factor of 4 for usual function folding.

It is expected that the size of the new PAE supporting Function Folding will increase by max. 25%. On the other hand 4 PAEs are reduced to 1.

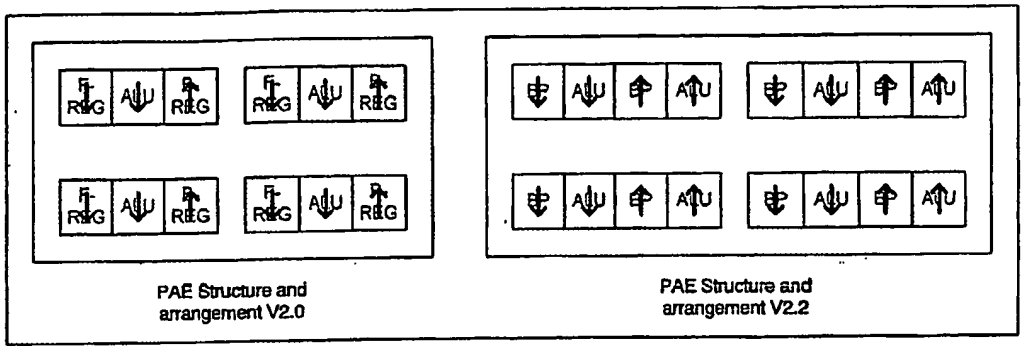
Assuming that in average not the optimum but only about 3 functions can be folded onto a single PAE a XPP64 could be replaced by a XPP21. Taking the larger PAEs into account the functionality of a XPP64 V2.0 should be executable on a XPP V2.2 with an area of less than half.

1.3 Array Structure

The V2.0 structure of the PAEs consumes much area for FREG and BREG and their associated bus interfaces. In addition feed backs through the FREGs require the insertion of registers into the feedback path, which result not only in an increased latency but also in a negative impact onto the throughput and performance of the XPP.

A new PAE structure and arrangement is proposed with the expectation to minimize latency and optimize the bus interconnect structure to achieve an optimized area.

The V2.2 PAE structure does not include BREGs any more. As a replacement the ALUs are alternating flipped horizontally which leads to improved placement and routing capabilities especially for feedback paths i.e. of loops.
Each PAE contains now two ALUs and two BP paths, one from top to bottom and one flipped from bottom to top.



1.4 Bus modifications

Within this chapter are optimizations described which reduce the required area and the amount of busses. However, this modifications are only proposals yet, since the have to be evaluated based on real algorithms. It is planed to compose a questionnaire to collect the necessary input from the application programmes.

1.4.1 Next neighbour

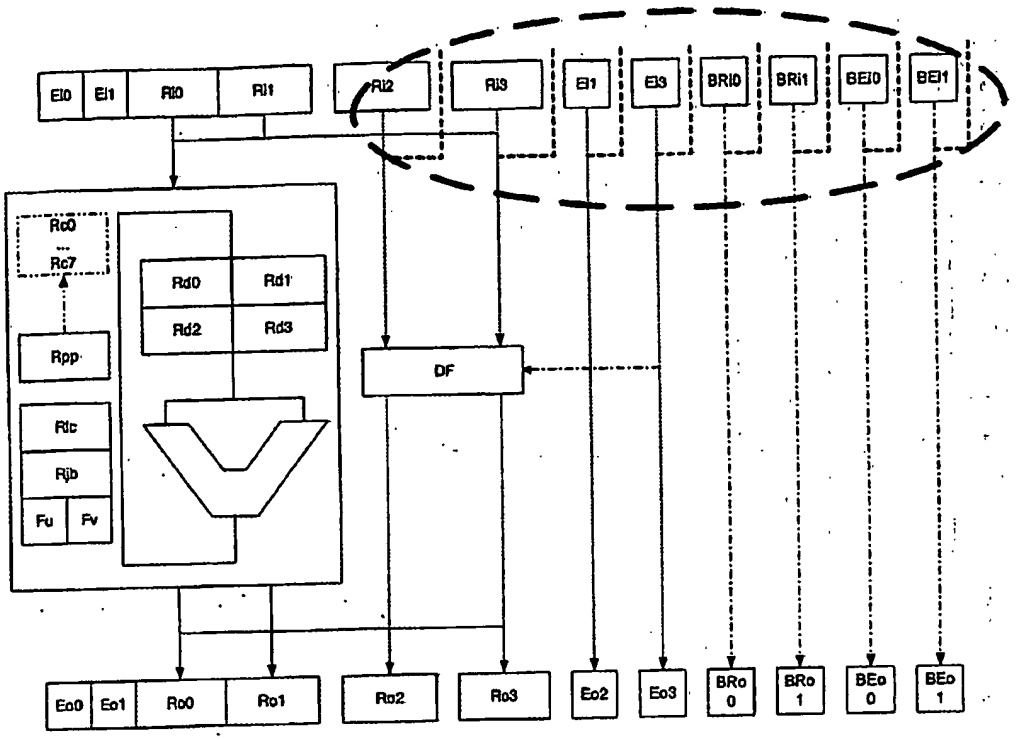
In V2.0 architecture a direct horizontal data path between two PAEs block a vertical data bus. This effect increases the required vertical busses within a XPP and drives cost unnecessarily.

Therefore in V2.2 a direct feed path between horizontal PAEs is proposed.

1.4.2 Removal of registers in busses

In V2.0 are registers implemented in the vertical busses which can be switched on by configuration for longer paths. This registers can furthermore be preloaded by configuration which requires a significant amount of silicon area.

It is proposed not to implement registers in the busses any more, but to use an enhanced DF or Bypass (PB) part within the PAEs which is able to reroute a path to the same bus using the DF or BP internal registers instead:



It is to evaluate

a) how many resources are saved for the busses and how many are needed for the PAEs

b) how often must registers be inserted, are 1 or max. 2 paths enough per PAE (limit is two since DF/BP offers max. 2 inputs)

1.4.3 Shifting n:1, 1:n capabilities from busses to PAEs

In V2.0 n:1 and 1:n transitions are supported by the busses which requires a significant amount of resources i.e. for the sample-and-hold stage of the handshake signals.

Depending on the size of n two different capabilities are provided with the new PAE structure:

$n \leq 2$	The required operations are done within the DF path of the PAE
$2 \leq n \leq 4$	The ALU path is required since 4 ports are necessary
$n > 4$	Multiple ALUs have to be combined

This method saves a significant amount of static resources in silicon but requires dedicated PAE resources at runtime.

It is therefore to evaluate

c) how much silicon area is saved per bus

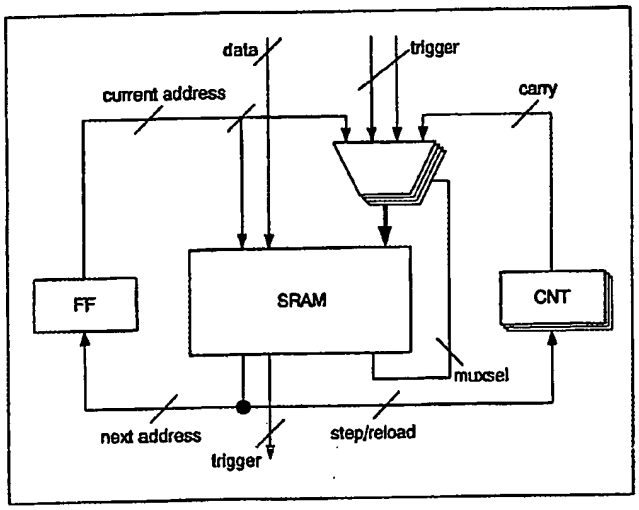
d) how often occurs n=2, $2 \leq n \leq 4$, $n > 4$

e) *the ratio between saved silicon area and required PAE resources*

1.5 FSM in RAM-PAEs

In the V2.0 architecture implementing control structures is very costly, a lot of resources are required and programming is quite difficult.

However memories can be used for a simple FSMs implementation. The following enhancement of the RAM-PAEs offers a cheap and easy to program solution for many of the known control issues, including HDTV.

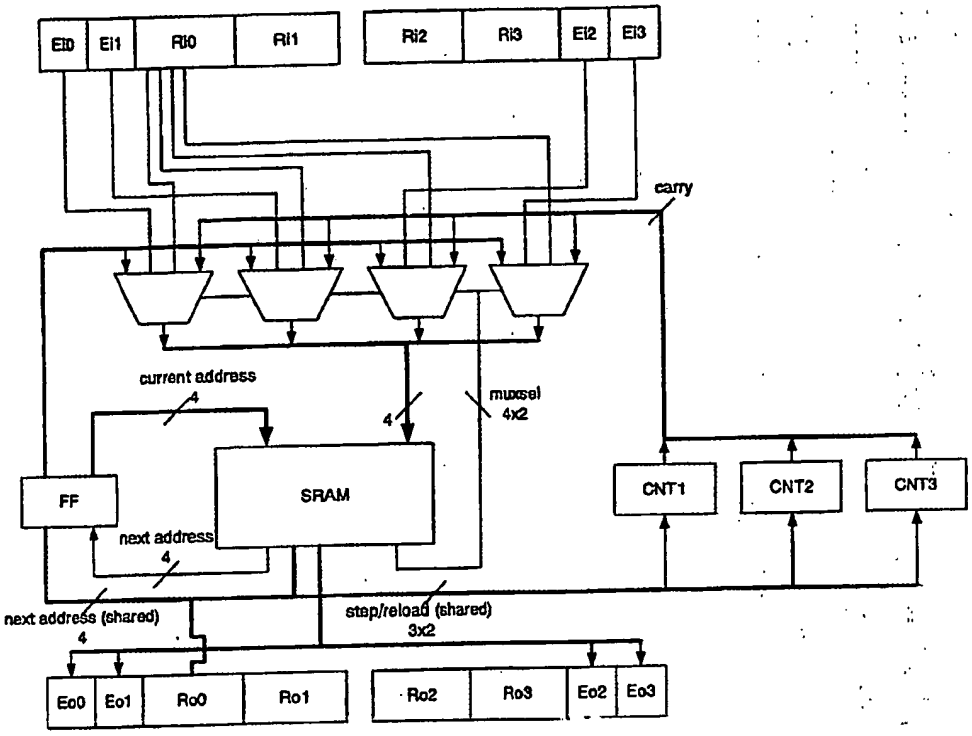


Basically the RAM-PAE is enhanced by an feedback from the data output to the address input through a register (FF) to supply subsequent address within each stage. Furthermore additional address inputs from the PAE array can cause conditional jumps, data output will generate event signals for the PAE array. Associated counters which can be reloaded and stepped by the memory output generate address input for conditional jumps (i.e. end of line, end of frame of a video picture).
At typical RAM-PAE implementation has about 16-32 data bits but only 8-12 address bits. To optimize the range of input vectors it is therefore suggestive to insert some multiplexers at the address inputs to select between multiple vectors, whereas the multiplexers are controlled by some of the output data bits.

The implementation for a XPP having 24bit wide data busses is sketched in the next figure. 4 event inputs are used as input, as well as the lower for bits of input port Ri0. 3 counters are implemented, 4 events are generated as well as the lower 10 bits of the Ro0 port.

The memory organisation is as follows:

8 address bits
24 data bits (22 used)
- 4 next address
- 8 multiplexer selectors
- 6 counter control (shared with 4 additional next address)
- 4 output

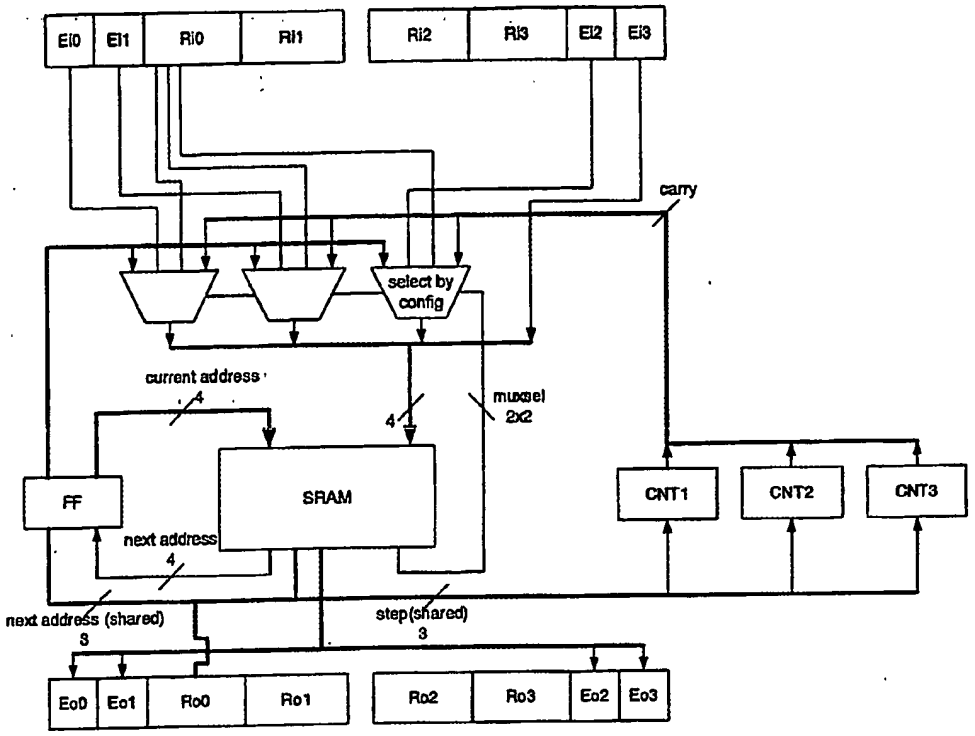


Please not that the typical memory mode of the RAM-PAE is not sketched in the block diagram above.

The width of the counters is according to the bus width of the data busses.

For a 16 bit implementation it is suggested to use the carry signal of the counters as their own reload signal (auto reload), also some of the multiplexers are not driven by the memory but "hard wired" by the configuration.

The proposed memory organisation is as follows:
- 8 address bits
- 16 data bits (16 used)
  - 4 next address
  - 4 multiplexer selectors
  - 3 counter control (shared with 3 additional next address)
  - 4 output



Actually the RAM-PAEs are not scaleable any more since the 16-bit implementation is different from the 24-bit implementation. It is to decide whether the striped down 16-bit implementation is used for 24-bit also.

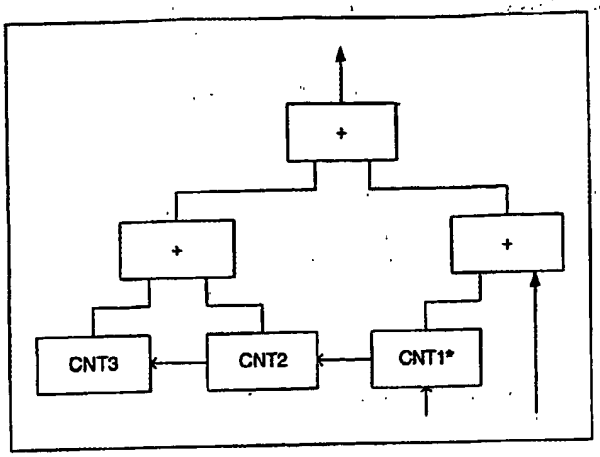
1.6 IOAG interface

1.6.1 Address Generators and bit reversal addressing

Implemented within the IO interfaces are address generators to support 1 to 3 dimensional addressing directly without any ALU-PAE resources. The address generation is done by 3 counters, each of them has configurable base address, length and step width.
The first counter (CNT1) has a step input to be controlled by the array of ALU-PAEs. Its carry is connected to the step input of CNT2, which carry again is connected to the step input of CNT3.
Each counter generates carry if the value is equal to the configured length. Immediately with carry the counter is reset to its configured base address.

One input is dedicated for addresses from the array of ALU-PAEs which can be added to the values of the counters. If one or more counters are not used they are configured to be zero.

In addition CNT1 supports generation of bit reversal addressing by supplying multiple carry modes.



1.6.2 Support for different word width

In general it is necessary to support multiple word width within the PAE array. 8 and 16 bit wide data words are preferred for a lot of algorithms i.e. graphics. In addition to the already described SIMD operation, the IOAG allows the split and merge of such smaller data words.

Since the new PAE structure allows 4 input and 4 output ports, the IOAG can support word splitting and merging as follows:

I/O 0	I/O 1	I/O 2	I3
16/24/32-bit data word			address
16-bit data word	16-bit data word		address
8-bit data word	8-bit data word	8-bit data word	address

Input ports are merged within the IOAG for word writes to the IO.
For output ports the read word is split according to the configured word width.

1.7 XPP / µP coupling

For a closed coupling of a µP and a XPP a cache and register interface would be the preferable structure for high level tools like C-compilers. However such a close coupling is expected not to be doable in a very first step.

Two different kind of couplings are necessary for a tight coupling:

a) memory coupling for large data streams: The most convenient method with the highest performance is a direct cache coupling, whereas an AMBA based memory coupling will be sufficient for the beginning (to be discussed with ATAIR)

b) register coupling for small data and irregular MAC operations: Preferable is a direct coupling into the processors registers with an implicit synchronisation in the OF-stage of the processor pipeline. However coupling via load/store- or in/out-commands as external registers is acceptable with the penalty of a higher latency which causes some performance limitation (already agreed with ATAIR)

2 Specification of ALU-PAE

2.1 Overview

The ALU-PAE comprises 3 paths:

- ALU arithmetic, logic and data flow handling
- DF data flow handling and bypass
- BP bypass

Each of the paths contains 2 data busses and 1 event bus. The busses of the DF path can be rerouted to the ALU path by configuration.

2.2 ALU path Registers

The ALU path comprises 12 data registers:

- Ri0-3 Input data register 0-3 from bus
- Rv0-3 Virtual output data register 0-3 to bus
- Rd0-3 Internal general purpose register 0-3

- Ei0-3 Event input 0-3 from bus
- Ev0-3 Virtual event output register 0-3 to bus
- Fu, FvFlag u and v according to the V2.0 PAE

Note: Ri2 and Ri3 belong typically to the DF path, but can be allocated for the ALU by configuration.

Eight instruction registers are implemented, each of them is 16 bit wide according to the opcode format.

- Rc0-7 Instruction registers

Three special purpose registers are implemented:

- Rlc Loop Counter, configured by CM, not accessible through ALU-PAE itself. Will be decremented according to JL opcode. Is reloaded after value 0 is reached.
- Rjb Jump-Back register to define the number of used entries in Rc[0..7]. It is not accessible through ALU-PAE itself.
  If Rpp is equal to Rjb, Rpp is immediately reset to 0. The jump back can be bound to a condition i.e. an incoming event. If the condition is missing, the jump back will be delayed.
- Rpp Program pointer

2.3 Data duplication and multiple input reads

Since Function Folding can operate in a purely data stream mode as well as in a sequential mode (see 1.2) it is useful to support Ri reads in dataflow mode (single

read only) and sequential mode (multiple read). The according protocols are described below:

Each Input register Ri can be configured to work in one of two different modes:

Dataflow Mode
This is the standard protocol of the V2.0 implementation:
A data packet is taken read from the bus if the register is empty, an ACK handshake is generated. If the register is not empty ACK the data is not latched and ACK is not generated.
If the register contains data, it can be read once. Immediately with the read access the register is marked as empty. An empty register cannot be read.

Simplified the protocol is defined as follows:

RDY & empty	→ full
	→ ACK
RDY & full	→ notACK
READ & empty	→ stall
READ & full	→ read data
	→ empty

Please note: pipeline effects are not taken into account in this description and protocol.

Sequencer Mode
The input interface is according to the bus protocol definition: A data packet is taken read from the bus if the register is empty, an ACK handshake is generated. If the register is not empty ACK the data is not latched and ACK is not generated.
If the register contains data it can be read multiple times during a sequence. A sequence is defined from Rpp = 0 to Rpp = Rjb. During this time no new data can be written into the register. Simultaneously with the reset of Rpp to 0 the register content is cleared an new data is accepted from the bus.

Simplified the protocol is defined as follows:

RDY & empty	→ full
	→ ACK
RDY & full	→ notACK
READ & empty	→ stall
READ & full	→ read data
(Rpp == Rjb)	→ empty

Please note: pipeline effects are not taken into account in this description and protocol.

2.4 Data register and event handling

Data registers are directly addressed, each data register can be individually selected. Since a two address opcode form is used, register operations follow the rule *op* $r_a \leftarrow r_a, r_b$. An virtual output register is selected by adding *'out'* behind the opcode. The result will be stored in r_a and copied to the virtual output register r_v as well according to the rule *op out* $(r_v, r_a) \leftarrow r_a, r_b$.
Please note, accessing input and (virtual) output registers follow the rules defined in chapter 2.3.

Rotating Select
Under normal conditions data and events are read one time according to the principles of Petri-Nets. Therefore for most applications a one time access per *Cycle* is sufficient. Also per definition one data or event is generated by a Petri-Net per channel and *Cycle*.

If Function Folding is done in a sequential manner synchronisation is achieved by using WAIT and SKIP commands. If multiple accesses to an event are required it can be copied by the READE instruction to the u or v flags which can be used successively for multiple commands.

The Rotating Select starts on the first access to events with the event E0, steps with the second access over E1 and E2, to E3 (at the fourth access) and restarts with the fifth access at E0 again.

Reset or Rpp == Rjb	after 1st event access	after 2nd event access	after 3rd event access	after 4th event access
E0	E1	E2	E3	continue with E0

Rotating select is supported for reading events and writing events with an explicit rotation counter for each read and write. Writing to events copies the value to the u flag at the same time, et(v) and ee(v) causes copying to the v flag.

For each opcode E0 and the internal flags u and v can be selected explicitly by the following selection modes. E0 can therefore be easy used as for multiple write event accesses per *Cycle* since there is no need to use the rotating select mode for E0 for most of the opcodes:

et (event target)
es (event source)

00	Internal u
01	Internal v
10	External Ev0
11	Rotating select: External next (Ev0/Ev1/EV2.0/EV2.2) and internal u flag

eventt (event target)
events (event source)

000	Internal u
001	Internal v
010	Ev0
011	Ev1
100	EV2.0
101	EV2.2
110	
111	Rotating select: External next

	(Ev0/Ev1/EV2.0/EV2.2) and internal u flag

Event Enable enables or disables writing a flag to an virtual event output. However the flag will be set in the internal u or v register anyhow.

ee (event enable)

0	Internal v or u
1	Internal v or u & Rotating select: External next (Ev0/Ev1/EV2.0/EV2.2) and internal v or u flag

Event sources
Instructions offering only ALU internal flags as source for the operations:

- SAT

The event addressing supports the selection between the u and v flag.

Instructions allowing directly addressed event sources using *eventt* and *events*:

- WAIT, SKIP, READE, WRITEE
- MERGE, DEMUX, SWAP

Instructions offering limited addressed event sources and rotating event select (*et*, *es*):

- SHL, SHR, DSHL, DSHR, DSHRU
- ADD, ADDC, SUB, SUBC

Event targets
Some instructions operate using rotating event select only (*et*, *es*):

- NOT, SORT, SORTU, CLZ, CLZU, AND, OR, XOR, EQ, CMP, CMPU

Some instructions support Event Enable only (*ee*):

- SHL, SHR, DSHL, DSHR, DSHRU
- ADD, ADDC, SUB, SUBC

2.4.1 n:1 Transitions

1:n transitions are not supported within the busses any more. Alternatively simple writes to multiple output registers Ro and event outputs Eo are supported. The Virtual Output registers (Rv) and Virtual Event (Ev) are translated to real Output registers (Ro) and real Events (Eo), whereas a virtual register can be mapped to multiple output registers.

To achieve this a configurable translation table is implemented for both data registers and event registers:

Rv Ev	Ro0 Eo0	Ro1 Eo1	Ro2 Eo2	Ro3 Eo3
0				
1				
2				
3				

Example:
Rv0 mapped to Ro0, Ro1
Rv1 mapped to Ro2
RV2.0 mapped to Ro3
RV2.2 unused

Rv	Ro0	Ro1	Ro2	Ro3
0	1	1	0	0
1	0	0	1	0
2	0	0	0	1
3	0	0	0	0

2.4.2 Accessing input and output registers (Ri/Rv) and events (Ei/Ev)

Independently from the opcode accessing input or output registers or events is defined as follows:

Reading an input register:

Register status	Operation
empty	wait for data
full	read data and continue operation

Writing to an output register:

Register status	Operation
empty	write data to register
full	wait until register is cleared and can accept new data

2.5 Opcode format

To achieve a small opcode size a two address code is used. The basic operation is:

$$op\ r_a \leftarrow r_a, r_b$$

Source registers can be Ri and Rd, target registers are Rv and Rd. A typical operation targets only Rd registers. If the source register for r_a is Ri[x] the target register will be Rd[x].
The translation is shown is the following table:

Target	Source r_a
Rd0	Rd0
Rd1	Rd1
Rd2	Rd2
Rd3	Rd3
Rd0	Ri0
Rd1	Ri1
Rd2	Ri2
Rd3	Ri3

Each operation can target a Virtual Output Register Rv by adding an *out* tag as a target identifier to the opcode:

$$\text{op out } r_a \leftarrow r_a, r_b$$

The transfer is now Ri[x] or Rd[x] to Rv[x] as shown in the table below:

Target	Source r_a
Rv0	Rd0
Rv1	Rd1
RV2.0	Rd2
RV2.2	Rd3
Rv0	Ri0
Rv1	Ri1
RV2.0	Ri2
RV2.2	Ri3

The opcode format is 16 bit wide, the standard formats are:

2.6 Clock

The PAE can operate at a configurable clock frequency of
- 1x Bus Clock
- 2x Bus Clock
- 4x Bus Clock
- [8x Bus Clock]

2.7 The DF path

The DataFlow path comprises the data registers Ri2&3 and Ro2&3 as well as the events Ei2&3 and Eo2&3. Each of the data registers Ri[n] is combined with an event E[n] whereas the according busses support different routings.

By configuration each data path and its associated event can be dedicated to the ALU path.

The DF path supports numerous instructions, whereas the instruction is selected by configuration and only one of them can be performed during a configuration, function folding is not available.

The following instructions are implemented:

1. ADD, SUB
2. NOT, AND, OR, XOR
3. SHL, SHR, DSHL, DSHR, DSHRU
4. EQ, CMP, CMPU
5. MERGE, DEMUX, SWAP
6. SORT, SORTU
7. ELUT

2.8 The BP path

The ByPass path is a simple horizontal network between the input data registers BRi0&1 and events BEi0&1 to the output registers BRo0&1 and events BEo0&1.

3 Input Output Address Generators (IOAG)

The IOAGs are located in the RAM-PAEs and share the same registers to the busses. An IOAG comprises 3 counters with forwarded carries. The values of the counters and an immediate address input from the array are added to generate the address. One counter offers reverse carry capabilities.

3.1 Adressing modes

Several addressing modes are supported by the IOAG to support typical DSP-like addressing:

Mode	Description
Immediate	Address generated by the PAE array
xD counting	Multidimensional addressing using IOAG internal counters xD means 1D, 2D, 3D
xD circular	Multidimensional addressing using IOAG internal counters, after overflow counters reload with base address
xD plus immediate	xD plus a value from the PAE array
Stack	decrement after "push" operations increment after "read" operations
Reverse carry	Reverse carry for applications such as FFT

3.1.1 Immediate Addressing

The address is generated in the array and directly fed through the adder to the address output. All counters are disabled and set to 0.

3.1.2 xD counting

Counters are enabled depending on the required dimension (x-dimensions require x counters). For each counter a base address and the step width as well as the maximum address are configured. Each carry is forwarded to the next higher and enabled counter; after carry the counter is reloaded with the start address.
A carry at the highest enabled counter generates an event, counting stops.

3.1.3 xD circular

The operation is exactly the same as for xD counting, with the difference that a carry at the highest enabled counter generates an event, all counters are reloaded to their base address and continue counting.

3.1.4 Stack

One counter (CNT1) is used to decrement after data writes and increment after data reads. The base value of the counter can either be configured (base address) or loaded by the PAE array.

3.1.5 Reverse carry

Typically carry is forwarded from LSB to MSB. Forwarding the carry to the opposite direction (reverse carry) allows generating address patterns which are very well suited for applications like FFT and the like. The carry is discarded at MSB.

For using reverse carry a value larger than LSB must be added to the actual value to count, wherefore the STEP register is used.

Example:
BASE = 0h
STEP = 1000b

Step	Counter Value
1	b0...00000
2	b0...01000
3	b0...00100
4	b0...01100
5	b0...00010
...	...
16	b0...01111
17	b0...00000

The counter is implemented to allow reverse carry at least for STEP values of -2, -1, +1, +2.

Appendix A
OpCodes

Notation:

Registers

name	explanation	number of bits		
target_r	Target register and related source register	2	Rd0	00
			Rd1	01
			Rd2	10
			Rd3	11
target_o	Target output register and related source register	2	Ro0	00
			Ro1	01
			Ro2	10
			Ro3	11
target	Target register and related source register. Target will be Rd or Ro (if target identifier is set)	3	Ri0	000
			Ri1	001
			Ri2	010
			Ri3	011
			Rd0	100
			Rd1	101
			Rd2	110
			Rd3	111
target_p	Target register pair and related source register pair	2	Ro0&1	00
			Ro2&3	01
			Rd0&1	10
			Rd2&3	11
source_i	Source input register	2	Ri0	00
			Ri1	01
			Ri2	10
			Ri3	11
source	Source register	3	Ri0	000
			Ri1	001
			Ri2	010
			Ri3	011
			Rd0	100
			Rd1	101
			Rd2	110
			Rd3	111
source_p	Source register pair	2	Ri0&1	00
			Ri2&3	01
			Rd0&1	10
			Rd2&3	11
r_pair_t	Source register and target register pair	2	target	source	
			Rd0&1	Ri0	00
			Rd2&3	Ri2	01

			Rd0&1 Rd0 10 Rd2&3 Rd2 11
tid	Target Identifier	1	0 Internal Register 1 Internal & External Register
val	Value	1	one bit value
valx	Value including don't care	2	00 0 01 1 10 X 11 X
val2	2 bit value	2	00 00 01 01 10 10 11 11
u/v	Select flag register Fu or Fv	1	0 Fu 1 Fv
et	event target	2	00 Internal u 01 Internal v 10 External E0 11 External next (E1/E2/E3)
es	event source	2	00 Internal u 01 Internal v 10 External E0 11 External next (E1/E2/E3)
event	event target (or source)	3	000 Internal u 001 Internal v 010 E0 011 E1 100 E2 101 E3 110 111 External next (E1/E2/E3)
ee	event enable	2	0 Internal 1 Internal & External next (E1/E2/E3)

0123456	7	8	9	10	11	12	13	14	15	IF	OF	Comment
NOP 000000	0	0	0	0	0	0	0	0	0			No Operation
READ 000000	0	target_r		0	source_i		0	1	0			Read packet from input port
WRITE 000000	1	target_o		0	source			1	0			Write packet to output port
MOVE 000000	0	target_r		1	source_r		0	1	0			Move data between register
LOAD 000000	1	target_r		1	0	0	0	1	0			Load register with constant
constant												
SAT 000000	0	target_r		0	0	0	1	0	0	U		Saturate if carry '0 if previous command was SUBC '1 if previous command was ADDC
SETUV 000000	0	val	u/v	0	0	1	1	0	0		U/V	Set Flags uf and vf
SWAPUV 000000	0	0	0	1	0	1	1	0	0		U/V	Swap u and v flag
NOT 000000	tid	target			1	et		0	0		U	
JR 000000	adr7							0	1			Jump relative
JL 000000	adr7							1	1			Jump relative if Rlc is not zero

MERGE 000001	tid	target_p	source_p	event					0	U		
DEMUX 000001	tid	target_p	source_p	event					1	U		
SWAP 000010	tid	target_p	source_p	event					0	U		
WAIT 000010	0	valx	0	0	event				1	U		Wait for incoming event
SKIP 000010	0	valx	0	1	event				1	U		Wait for incoming event
EOPTR 000010	0	val2	1	0	0	0	0	1				Set event output pointer
EIPTR												Set event input pointer

012345	6	7	8	9	10	11	12	13	14	15	IF	OF	Comment
SHL 100000	tid	r_pair_t		source			es		ee(u)	ee(v)	U	U/V	
SHR 100001	tid	r_pair_t		source			es		ee(u)	ee(v)	U	U/V	
DSHL 100010	tid	r_pair_t		source			es		ee(u)	ee(v)	U	U/V	
DSHR 100011	tid	r_pair_t		source			es		ee(u)	ee(v)	U	U/V	
DSHRU 101000	tid	r_pair_t		source			es		ee(u)	ee(v)	U	U/V	
ASI 101001 101010 101011 101100 101101 101110													Application specific instructions

01234	5	6	7	8	9	10	11	12	13	14	15	IF	OF	Comment
ADD 11000	tid	target			source			es		ee(u)	ee(v)	U	U/V	
ADDC 11001	tid	target			source			es		ee(u)	ee(v)	U	U/V	
SUB 11010	tid	target			source			es		ee(u)	ee(v)	U	U/V	
SUBC 11011	tid	target			source			es		ee(u)	ee(v)	U	U/V	

000010	0	val2		1	0	0	0	1	1				
READE 000010	0	0	u/v	1	1	event			1	U/V			Read Event to U/V
WRITEE 000010	0	1	u/v	1	1	event			1		U/V		Write Event from U/V
SORT 000011	tid	target_p		source_p		et(u)		et(v)			U/V		Sort two data packets
SORTU 000100	tid	target_p		source_p		et(u)		et(v)			U/V		Sort two unsigned data packets
CLZ 000101	tid	target		event		0	0				U		Count leading zeros
CLZU 000101	tid	target		event		0	1				U		Count leading zeros unsigned
AND 000110	tid	target		source		et(u)					U		
OR 000111	tid	target		source		et(u)					U		
XOR 001000	tid	target		source		et(u)					U		
EQ 001001	tid	target		source		et(v)					U		Equal
CMP 001001	tid	r_pair_t		source_p		et(u)		et(v)			U/V		
CMPU 001010	tid	r_pair_t		source_p		et(u)		et(v)			U/V		
BSHL 001011	tid	r_pair_t	0	source		0	0						Barrel Shift left
BSHR 001011	tid	r_pair_t	0	source		0	1						Barrel Shift right
BSHRU 001011	tid	r_pair_t	0	source		1	0						Barrel Shift right unsigned
MUL 001011	tid	r_pair_t	1	source		0	0						
MULU 001011	tid	r_pair_t	1	source		0	1						
DIV 001011	tid	r_pair_t	1	source		1	1						

Akte: PACT48/EP ----

European Patent Application

Applicant: PACT XPP Technologies AG
Muthmannstrasse 1
80939 München

Representative: European Patent Attorney
Claus Peter Pietruk
Heinrich-Lilienfein-Weg 5
D-76299 Karlsruhe
No. 0 085 850

Title: Device and method for data processing

Claims

1. A data processing unit having a plurality of cells, in particular coarse-grained logic cells, interconnected and/or interconnectable for data processing wherein at least one cell, preferably a number of cells have instruction storage means for storing instructions to be executed so as that said coarse-grained cells form a plurality of sequencers within said array.

2. A method for operating data processing in an array comprising a plurality of logic cells, in particular coarse-grained logic cells interconnected and/or interconnectable for data processing, wherein data are transferred into cells from an input and/or from other cells via busses, characterised in that at least some of the busses are used for effecting a configuration of said

cells, in particular during runtime and/or without effecting cells not to be configured.

3. Method according to claim 2, wherein said busses are used with a frequency different from the frequency of data processing in at least some of the cells.

-.-